# A Frequency Compensated Clock for Precision Synchronization using IEEE 1588 Protocol and its Application to Ethernet

Sivaram Balasubramanian, Kendal R. Harris and Anatoly Moldovansky
Rockwell Automation

## Abstract

In a distributed control system containing multiple clocks, individual clocks tend to drift apart. Hence, individual clocks need to be corrected to synchronize and maintain time to required accuracy. In this paper, we present a frequency compensated clock to achieve precision synchronization amongst distributed clocks using IEEE 1588 protocol to exchange timing information. Further, we explore its application to Ethernet.

## 1.0 Introduction

In a distributed control system containing multiple clocks, individual clocks tend to drift apart due to instabilities inherent in source oscillators and environmental conditions such as temperature, air circulation, mechanical stresses, vibration, aging etc. Hence, some kind of correction is necessary to synchronize individual clocks to maintain the notion of global time, which is accurate to some requisite clock resolution. The IEEE 1588 standard [1] for a precision clock synchronization protocol for networked measurement and control system standardizes time and defines several messages that can be used to distribute timing information, but leaves it to implementation the means to achieve precision clock synchronization.

One important implementation and application specific issue is what to do with distributed time. A simple solution is to snap to new time value when time distribution arrives by setting the local clock to new value. In distributed control and in many other applications, this simple solution is not practicable. For example, in distributed drive/motion control applications snapping to new time value may break those applications. Therefore, it is necessary to manage drifts and maintain time between time distributions. One way to achieve this is to use accurate temperature compensated or oven controlled oscillators as clock source. However these oscillators are expensive. Hence, a solution using inexpensive standard crystal oscillators is preferable.

Figure 1 shows the clock drift phenomenon on a slave clock between time distribution (Sync) messages from master clock. At time $\text{MasterSyncTime}_{n-1}$ the master clock sends Sync message to slave clock(s). The slave receives this message after a delay equal to transit time from master to slave (MasterToSlaveDelay) at $\text{MasterClockTime}_{n-1}$ and sets its clock such that

$$\text{SlaveClockTime}_{n-1} = \text{MasterClockTime}_{n-1} = \text{MasterSyncTime}_{n-1} + \text{MasterToSlaveDelay}$$

Where n denotes the Sync message count from master to slave.

At time $\text{MasterSyncTime}_n$ the master sends next Sync message to slave clock. The slave receives this message at $\text{MasterClockTime}_n$, but the $\text{SlaveClockTime}_n$ has drifted apart by SlaveClockDrift. Therefore

$$\text{MasterClockTime}_n = \text{MasterSyncTime}_n + \text{MasterToSlaveDelay}$$

$$\text{SlaveClockTime}_n = \text{MasterSyncTime}_n + \text{MasterToSlaveDelay} + \text{SlaveClockDrift}$$

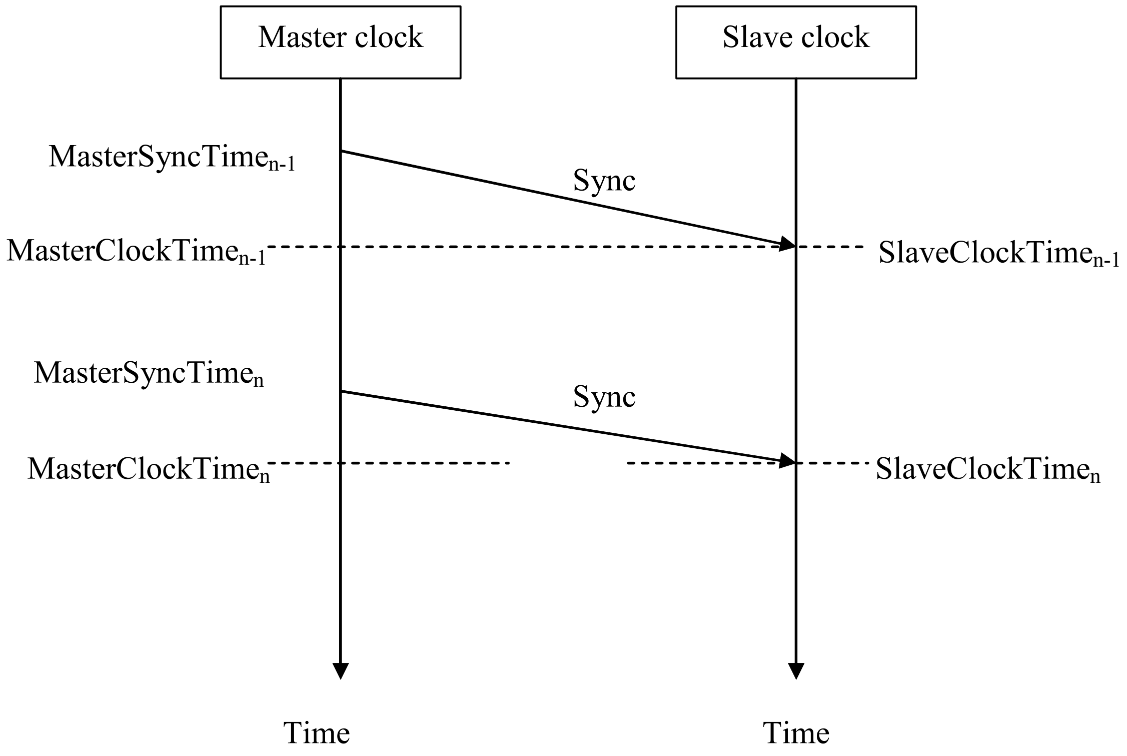

**Figure 1: Timing Messages**

Therefore, the slave clock has to be corrected such that

1. The frequencies of master and slave clocks have to equal each other.
2. The discrepancy between values of MasterClockTime and SlaveClockTime at any given instant should be eliminated or at least minimized.

The following sections present a simple but accurate, frequency compensated clock using standard inexpensive crystal oscillators to achieve these two objectives.

## 2.0 Frequency Compensated Clock

As shown in Figure 2, the frequency compensated clock is comprised of a p-bit clock counter, a q-bit accumulator as high precision frequency divider and an r-bit addend register holding frequency compensation value (FreqCompensationValue). All the constituents of frequency compensated clock operate at the frequency of input oscillator (FreqOscillator). The

FreqCompensationValue contained in addend register is added to accumulator once every 1/FreqOscillator. The clock counter is incremented whenever the accumulator asserts an overflow pulse signal. Therefore the nominal frequency at which the clock counter is incremented is FreqClock and is determined by FreqCompensationValue.

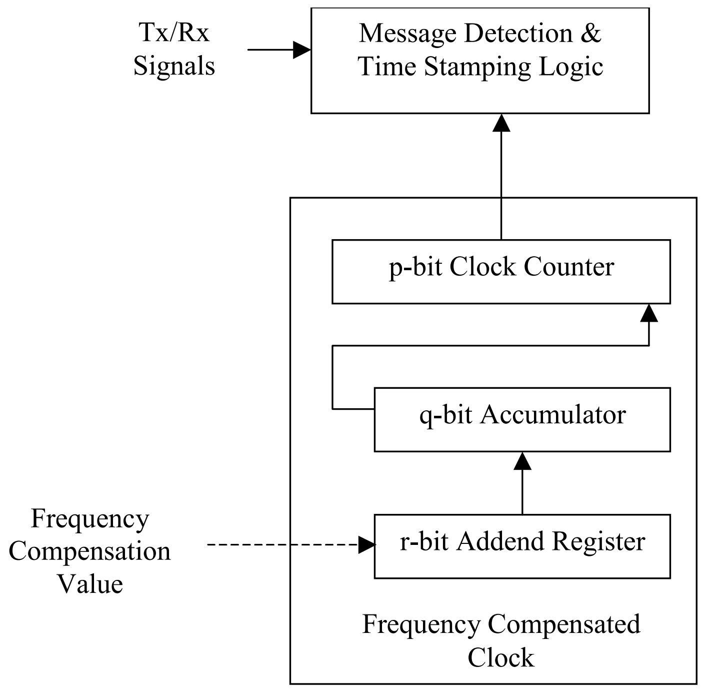
Tx/Rx Signals
Message Detection & Time Stamping Logic
p-bit Clock Counter
q-bit Accumulator
Frequency Compensation Value
r-bit Addend Register
Frequency Compensated Clock

**Figure 2: Frequency Compensated Clock**

The ratio of FreqOscillator to FreqClock is a design constant chosen initially to be a number greater than 1.0001, where 1.0001 corresponds to 0.01% oscillator stability. Though for 0.01% (+/-100 PPM) stability oscillator frequency drift rate may be an infinite precision real number between 0.9999 and 1.0001, it is not possible to compensate in hardware for infinite precision. Let CompensationPrecision be a constant number representing the highest precision desired for frequency compensation, and SyncInterval be a number representing interval between Sync messages in seconds. The following relationships can be used to determine the widths of accumulator, clock counter and addend register that will provide adequate precision for frequency compensation.

FreqDivisionRatio = FreqOscillator / FrequencyClock
CompensationPrecision <= 1 / (SyncInterval * FreqClock)
$2^q$ >= FreqDivisionRatio / CompensationPrecision
$2^r$ >= $2^q$ / FreqDivisionRatio
$2^p >= 2^q$

For example, FreqOscillator is 50MHz, FreqClock is 40 MHz, FreqDivisionRatio is 1.25, SyncInterval is 1 second, CompensationPrecision is $1 \times 10^{-9}$, width of accumulator q is 32, width of addend register r is 32 and width of clock counter p is 64 to provide a clock resolution of 25 nanoseconds.

The accumulator-addend register pair implements a high precision frequency divider whose accuracy is at least equal to CompensationPrecision. The accumulator-addend register pair operate by emitting N clock cycles of width Floor(FreqDivisionRatio) / FreqOscillator followed by a cycle with a width of Floor(FreqDivisionRatio+1) / FreqOscillator or Floor(FreqDivisionRatio-1) / FreqOscillator as appropriate. Where Floor operator rounds down real number to an integer and N is an integer determined by FreqCompensationValue.

**3.0 Frequency Compensation Algorithm**

Initially the slave clock is set with a $\text{FreqCompensationValue}_0$ as follows

$\text{FreqCompensationValue}_0 = 2^q / \text{FreqDivisionRatio}$

Let the MasterToSlaveDelay be set to zero initially and the algorithm described below be applied. After a few Sync cycles, frequency lock will occur. Then the slave clock can determine precise value for MasterToSlaveDelay and resynchronize with master using new value.

The following is a description of algorithm proper. At time $\text{MasterSyncTime}_n$ the master sends a Sync message to slave clock. The slave receives this message when its local clock is $\text{SlaveClockTime}_n$ and computes $\text{MasterClockTime}_n$ as

$\text{MasterClockTime}_n = \text{MasterSyncTime}_n + \text{MasterToSlaveDelay}$

The master clock count for current Sync cycle, $\text{MasterClockCount}_n$ is given by

$\text{MasterClockCount}_n = \text{MasterClockTime}_n - \text{MasterClockTime}_{n-1}$

The slave clock count for current Sync cycle, $\text{SlaveClockCount}_n$ is given by

$\text{SlaveClockCount}_n = \text{SlaveClockTime}_n - \text{SlaveClockTime}_{n-1}$

The difference between master and slave clock counts for current Sync cycle, $\text{ClockDiffCount}_n$ is given by

$\text{ClockDiffCount}_n = \text{MasterClockTime}_n - \text{SlaveClockTime}_n$

The frequency-scaling factor for slave clock, $\text{FreqScaleFactor}_n$ is given by

$\text{FreqScaleFactor}_n = (\text{MasterClockCount}_n + \text{ClockDiffCount}_n) / \text{SlaveClockCount}_n$

The frequency compensation value for addend register, $\text{FreqCompensationValue}_n$ is given by

$\text{FreqCompensationValue}_n = \text{FreqScaleFactor}_n * \text{FreqCompensationValue}_{n-1}$

Theoretical result of this algorithm is that at $\text{SlaveClockTime}_{n+1}$, the frequencies of master and slave clocks would lock, and differences between their clocks $\text{ClockDiffCount}_{n+1}$ would be zero. In other words, this algorithm would achieve a lock in one Sync cycle. Therefore the following statements are true with respect to this algorithm, assuming constant network propagation delays and gradually changing operating conditions such as temperature:

1. When the clocks drift at a constant rate, the frequency and clock locks between master and slave will be achieved in one Sync cycle and would stay locked after that.
2. When the clocks drift apart at a gradually changing drift rate over time, the slave clock will track master clock very closely by locking frequencies/clock values in one Sync cycle whenever clock drift is detected.

This algorithm has a self-correcting property. If for any reason the slave clocks set their initial local clock values from master slightly incorrect, it will be corrected at a cost of couple of more Sync cycles.

## 4.0 Implementation on Ethernet

The frequency compensated clock was implemented on an FPGA with message detection and time stamping logic monitoring media independent interface signals between network transceiver and medium access controller for IEEE 802.3/Ethernet. An inexpensive standard crystal oscillator with a frequency of 50MHz was used as input. The oscillator frequency was divided down to a nominal clock frequency of 40MHz or 25ns clock resolution. The slave clock was locked on to master clock with a deviation of +/-25ns most of the time and a worst case deviation of +/-100ns was observed on a switched 100Mbps Ethernet with some background traffic. The behavior was similar in the case of 100Mbps half-duplex mode network with repeater.

The accuracy of synchronization was largely limited by the random delay components introduced by network transceivers on transmit/receive ends. With higher background traffic switch delays will further limit accuracy and will necessitate some sort of filtering. In addition random delays tend to increase the number of Sync cycles required to achieve lock. One important requirement for frequency compensated clock is to have a source oscillator with good short term stability for 1 sec, since it cannot compensate for that. The Allen deviation of many standard crystal oscillators were found to be $50\text{x}10^{-9}$ for 1 second gate time and some were found to be even better at couple of parts per billion. Hence the frequency compensated clock with standard crystal oscillators can satisfy accuracy requirements for many applications.

## 6.0 Conclusions

The frequency compensated clock provides a simple and accurate way to implement IEEE 1588 standard for precision synchronization. It provides high accuracy with inexpensive standard crystal oscillators. It maintains time and manages drift accurately even in the presence of lost synchronization messages. It can be easily adapted for various networks and applications.

## 7.0 References

1. IEEE Std 1588-2002, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 8 November 2002.
2. Esker, Lawrence W., Low Deviation Synchronization Clock, U.S. Patent 6,236,277. September 30, 1999.
3. Schossmaier Klaus, Schmid Ulrich, Horauer Martin, Loy Dietmar, Specification and Implementation of the Universal time Coordinated Synchronization Unit (UTCSU), Journal of Real-Time Systems, vol. 12, no.3, pp.295-327, May 1997.